A Bayesian Approach to Predicting NFL Quarterback Scores in Fanduel Tournaments

STAT 578, Fall 2017, ***Team 5****: Aaron Ray, Kiomars Nassiri, Michael Chan*

October 25, 2017

Project Description

The National Football League (NFL), being one of the major professional sports leagues in North America, has a wide audience. participates in the NFL craze by competing in fantasy football tournaments organized by the daily fantasy site, "FanDuel.com". Participants in a **Fantasy Football** game act as the managers of a virtual footbal team and try to maximize their points by picking up the best line-up. Points are given based on actual performance of players in real-world competition. For the purpose of this project we have chosen to work with the data gathered from the **FanDuel** internet company. We will leverage a Hierarchical Bayesian approach with the Markov Chain Monte Carlo method to predict the fantasy points likely to be scored by an NFL quarterback in any given game. The goal is to predict the points scored by each player given certain prior conditions and predictor variables that will assist our model in providing credible posterior prediction intervals.

The analysis is inspired by the study presented in the article, **Bayesian Hierarchical Modeling Applied to Fantasy Football Projections for Increased Insight and Confidence**, by Scott Rome.

Team Members

- **Aaron Ray** (aaronwr2@illinois.edu)*
- **Kiomars Nassiri** (nassiri2@illinois.edu)
- **Michael Chan** (mhchan3@illinois.edu)

*Contact Person

Dataset Description

Team has set up a process to gather the historical data from the RotoGuru website. The following is the code used to get the data from RotoGuru:

```
# Scrape rotoguru1 site for weekly FanDuel stats and bind each week's data to the
# pre-defined dataframe, 'd'.

for(year in 2014:2017){
  for(week in 1:16){
    page = read_html(
      gsub(" ","",
           paste("http://rotoguru1.com/cgi-bin/fyday.pl?week=",week,"&year=",
                 year ,"&game=fd&scsv=1")
      ))
    dtext = page %>% html_nodes("pre") %>% html_text(trim = TRUE)
    dtable = read.table(text=dtext, sep = ";", header=TRUE, col.names = cnames,
```

```
                        quote=NULL)
    d = rbind(d,dtable)
  }
}
```

Data cleaning is performed using R routines. Some data cleaning tasks are needed to calculate Player rank.

Response Variables

- `FanDuelPts`: Points position at the end of a single game

Predictor Variables

- `AvgPts5Wks`: The 5 game average points of the player
- `AvgOppPAP7Wks` : The 7 game average Opposing Points Allowed to Position (OppPAP) by the current player's opposing defense. For example, if the Buffalo Bills defense allowed a total of 30 points per game to wide receivers for six games straight, then this number would equal to the average of 30 for any wide receiver facing the Bills defense.
- `Position`: The position the player plays
- `HomeGame`: Whether it is home game.
- `Rank`: The rank of a player based on recent performance

Analysis Ideas

Model

At the lowest level, we model the performance (`FanDuelPts`) as normally-distributed around a true value:

$y|\alpha, \beta_{defense}, \beta_{home}, \beta_{away}, \sigma_r^2 \sim N(\alpha + X_{defense}.\beta_{defense} + X_{home}.\beta_{home} + X_{away}.\beta_{away}, \sigma_y^2 I)$

where

α = The average fan duel point of the previous 5 weeks of the player, `AvgPts5Wks`

$\beta_{defense,p}$ = defense coefficient against team t for position p

$\beta_{home,p,r}$ = home coefficient for position p and a rank r player

$\beta_{away,p,r}$ = Away coefficient for position p and a rank r player

y = `FanDuelPts`

x_p = interaction indicator term for opposing team score allowed by position p

$x_{home,p,r}$ = interaction indicator term for rank r, position p, and whether it is home game

At higher level, we model the defense effect, $\beta_{defense}$, as how good(bad) a particular team's defense is against the player's position. We pool the effect based on the position of the player. That is, the defense coefficient is normally distributed from the same position specific distribution.

$\beta_{defense,p} \sim N(\delta_p, \sigma_\delta^2)$

where σ_δ is constant = 1000

For the home and away game effect, β_{home} and β_{away}, we model the effect for player of the same rank has the same distribution. We model the home and away game effect to be the same for players of the same position.

$\beta_{home,p,r} \sim N(\eta_r, \sigma_\eta^2)$

$\beta_{away,p,r} \sim N(\rho_r, \sigma_\rho^2)$

where σ_η, σ_ρ are constant $= 1000$

We will approximate non informative prior using:

$\sigma_y \sim Inv - gamma(0.0001, 0.0001)$

$\delta \sim N(0, 10000^2)$

$\eta \sim N(0, 10000^2)$

$\rho \sim N(0, 10000^2)$

Here is the JAGS model:

```
#sink("fdp.bug")
#cat("
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(alpha[i] + inprod(X.defense[i, ], beta.defense)
                 + inprod(X.home[i, ], beta.home)
                 + inprod(X.away[i, ], beta.away), sigmasqinv)
  }

  # The entry of the beta.defense corresponds to Opponent:Position
  # In our model, we pool the beta.defense based on position.
  # i.e. All defense effects of the same position are drawn from the same distribution
  for (p in 1:Num.Position) {
    beta.defense[p] ~ dnorm(delta[p], 1/1000^2)
    delta[p] ~ dnorm(0, 1/100000^2)
  }

  # The entry of the beta.home and beta.away corresponds to Rank:Position
  # In our model, we pool the beta.home/away based on rank
  for (r in 1:Num.Rank) {
    for (t in 1:Num.Position) {
      beta.home[(t-1) * Num.Rank + r] ~ dnorm(eta[r], 1/1000^2)
      beta.away[(t-1) * Num.Rank + r] ~ dnorm(rho[r], 1/1000^2)
    }
    eta[r] ~ dnorm(0, 1/100000^2)
    rho[r] ~ dnorm(0, 1/100000^2)
  }

  sigmasqinv ~ dgamma(0.0001, 0.0001)
  sigmasq <- 1/sigmasqinv
}
#    ",fill = TRUE)
#sink()
```

Sample Data

```
fdp <- read.csv("fdpfinal.csv", sep = ',', header = TRUE)

head(fdp)
```

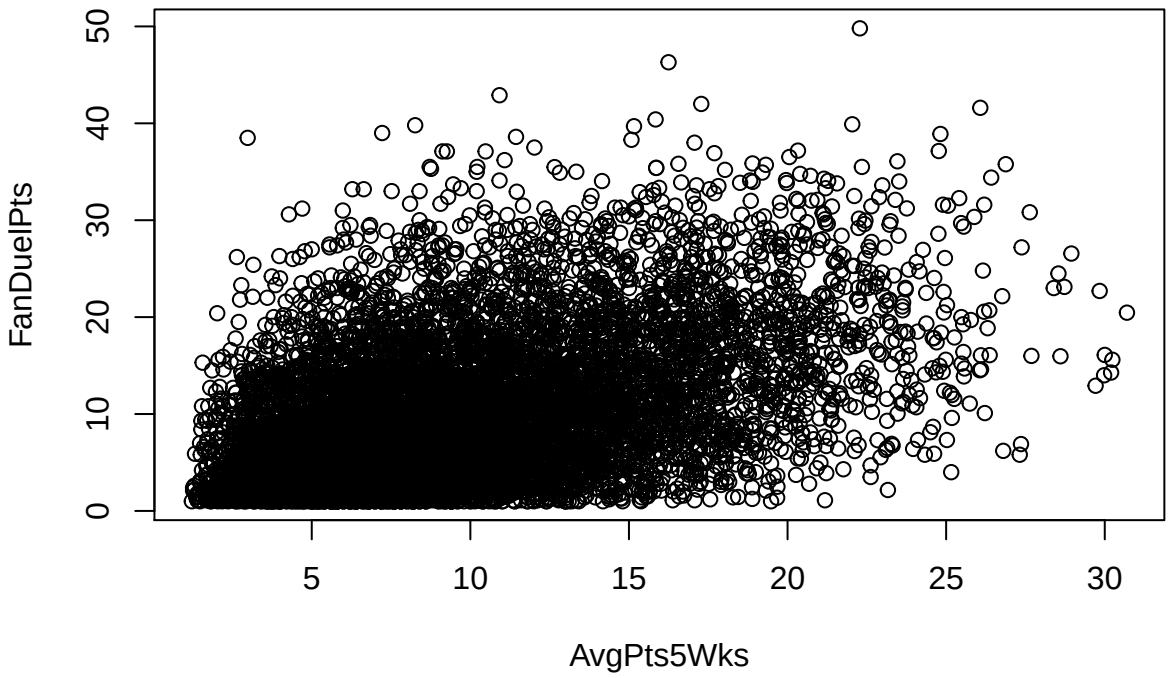
```
##   Position Year YearWeek Opponent Week PlayerId             Name    Team
## 1       QB 2015   201513 Steelers   13     1060 Hasselbeck, Matt   Colts
## 2       QB 2015   201514  Jaguars   14     1060 Hasselbeck, Matt   Colts
## 3       QB 2015   201515   Texans   15     1060 Hasselbeck, Matt   Colts
## 4       QB 2015   201516 Dolphins   16     1060 Hasselbeck, Matt   Colts
## 5       QB 2015   201501   Ravens    1     1081  Manning, Peyton Broncos
## 6       QB 2015   201502   Chiefs    2     1081  Manning, Peyton Broncos
##   HomeGame FanDuelPts FanDuelSalary AvgOppPAP7Wks SdOppPAP7Wks OallAvgPAP
## 1        0       6.86          6500         20.95        8.045      17.44
## 2        0       9.08          6600         24.16        6.738      17.44
## 3        1       8.98          6400         16.36        9.690      17.44
## 4        0       3.96          6000         20.46        6.002      17.44
## 5        1       5.90          9100         18.99       11.697      17.44
## 6        0      21.24          8200         13.85        4.342      17.44
##   OallStdevPAP AvgPts5Wks StdevPts5Wks OffRnk5Wks DefRnk7Wks
## 1        2.909      14.85        4.824      Rank4      Rank1
## 2        2.909      14.82        4.897      Rank4      Rank1
## 3        2.909      13.56        5.490      Rank4      Rank3
## 4        2.909      12.11        5.566      Rank4      Rank2
## 5        2.909      14.96        8.496      Rank3      Rank1
## 6        2.909      10.53        5.011      Rank4      Rank4
```

```
fdp['Rank'] = fdp$OffRnk5Wks
fdp['Locality'] = 'Away'
fdp[fdp$HomeGame == 1, 'Locality'] = 'Home'
```

Simple Ideas

$y|\alpha \sim N(\alpha, \sigma_y^2 I)$

```
mod.classic = lm(FanDuelPts ~ AvgPts5Wks, data = fdp)

plot(FanDuelPts ~ AvgPts5Wks, data = fdp)
```



All over the place, let's add the team defense

`X.defense` is the indicator matrix

** Set up train data **

```
fdp_train=fdp[fdp$Year == 2015  & ((fdp$Position == "QB"& fdp$FanDuelSalary > 6500 & !is.na(fdp$FanDuel
#fdp_train=fdp[fdp$Year == 2015 & (fdp$Position == "QB" | fdp$Position == "RB") & fdp$FanDuelSalary > 6
fdp_train = droplevels(fdp_train)

Num.Opponent = length(unique(fdp_train[, "Opponent"]))
Num.Position = length(unique(fdp_train[, "Position"]))
Num.Rank = length(unique(fdp_train[, "Rank"]))

if (Num.Position == 1) {
  #X.offense = model.matrix(~ 0 + AvgPts5Wks, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks, data=fdp_train)
  X.home = model.matrix(~ 0 + Rank , data=fdp_train)
  X.away = model.matrix(~ 0 + Rank , data=fdp_train)
} else {
  #X.offense = model.matrix(~ 0 + AvgPts5Wks:Position, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks:Position, data=fdp_train)
  X.home = model.matrix(~ 0 + Rank:Position , data=fdp_train)
  X.away = model.matrix(~ 0 + Rank:Position , data=fdp_train)
}
```

```
X.home = X.home * fdp_train$HomeGame
X.away = X.away * (1- fdp_train$HomeGame)
X = cbind(X.defense, X.home, X.away)
```

```
library(rjags)
set.seed(20171008)
```

```
# Initialization List for the 4 chains
jags.inits=list(
  list( sigmasqinv=    0.01,  delta = rep(-100000, Num.Position),
        eta = c(100000, -100000, 100000, -100000)[1:Num.Rank],
        rho = c(-100000, 100000, -100000, 100000)[1:Num.Rank],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 ),
  list( sigmasqinv=    0.01,  delta = rep(100000, Num.Position),
        eta = c(100000, -100000, -100000, 100000)[1:Num.Rank],
        rho = c(-100000, 100000, 100000, -100000)[1:Num.Rank],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 1 ),
  list( sigmasqinv=0.000001,  delta = rep(-100000, Num.Position),
        eta = c(-100000, 100000, -100000, 100000)[1:Num.Rank],
        rho = c(100000, -100000, 100000, -100000)[1:Num.Rank],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 2 ),
  list( sigmasqinv=0.000001,  delta = rep(100000, Num.Position),
        eta = c(-100000, 100000, 100000, -100000)[1:Num.Rank],
        rho = c(100000, -100000, -100000, 100000)[1:Num.Rank],
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 3 )
)

data.jags <- list(
  y= fdp_train$FanDuelPts,
  alpha = fdp_train$AvgPts5Wks,
  X.defense = X.defense,
  X.home = X.home,
  X.away = X.away,
  Num.Position=Num.Position,
  #Num.Opponent=Num.Opponent,
  Num.Rank=Num.Rank
)
```

```
burnAndSample = function(m, N.burnin, N.iter, show.plot, mon.col) {
  update(m, N.burnin) # burn-in

  x <- coda.samples(m, mon.col, n.iter=N.iter)

  if(show.plot) {
    plot(x, smooth=FALSE)
  }

  gelman.R = gelman.diag(x, autoburnin=FALSE, multivariate = FALSE)
  print(gelman.R)

  result <- list(
    coda.sam = x,
    gelman.R.max=max(gelman.R$psrf[, 1])
```

```
  )

  return(result)
}
```

```
runModel=TRUE
runSample=TRUE

mon.col <- c("delta", "eta", "rho", "beta.defense", "beta.home", "beta.away", "sigmasq")

NSim = 30000
if (runModel) {
  m <- jags.model("fdp.bug", data.jags, inits = jags.inits, n.chains=4, n.adapt = 1000)
  save(file="fdp.jags.model.Rdata", list="m")
} else {
  load("fdp.jags.model.Rdata")
  m$recompile()
}
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 781
##    Unobserved stochastic nodes: 29
##    Total graph size: 19177
##
## Initializing model
```

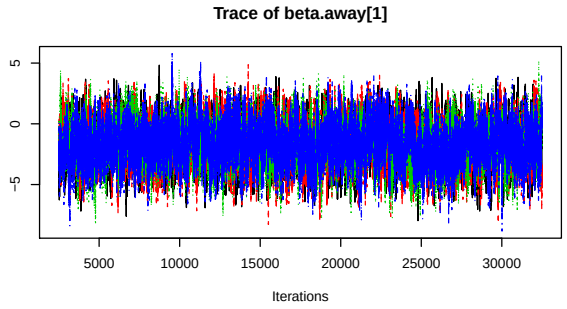
```
load.module("dic")
```

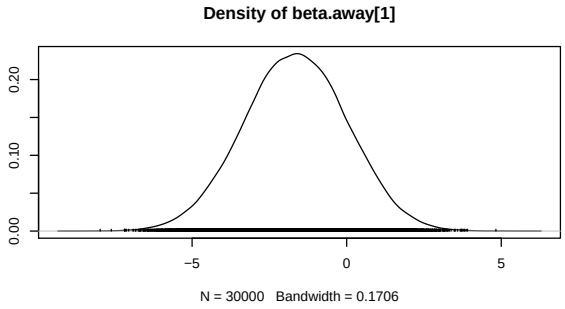
```
## module dic loaded
```

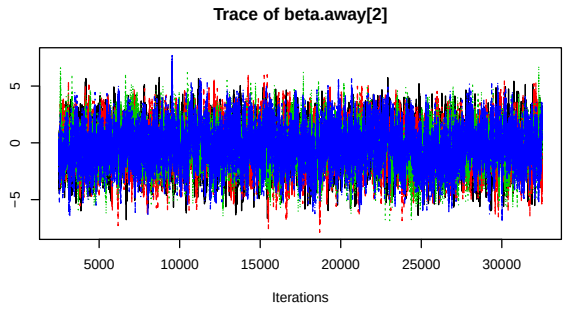
```
N.Retry.Loop = 1
if (runSample) {
  N.burnin=2500/2
  for (loopIdx in 1:N.Retry.Loop) {
    (start_time <- Sys.time())
    (N.burnin = N.burnin * 2)
    result = burnAndSample(m, N.burnin, NSim, show.plot=FALSE, mon.col = mon.col)
    (end_time <- Sys.time())
    (result$gelman.R.max)
  }
  save(file=paste("fdp.jags.samples.", N.burnin, ".Rdata", sep=""), list="result")
  save(file=paste("fdp.jags.model.", N.burnin, ".Rdata", sep=""), list="m")
} else {
  N.burnin=2500/2 * (2**N.Retry.Loop)
  load(paste("fdp.jags.samples.", N.burnin, ".Rdata", sep=""))
  load(paste("fdp.jags.model.", N.burnin, ".Rdata", sep=""))

  gelman.diag(result$coda.sam, autoburnin=FALSE, multivariate = FALSE)
}
```

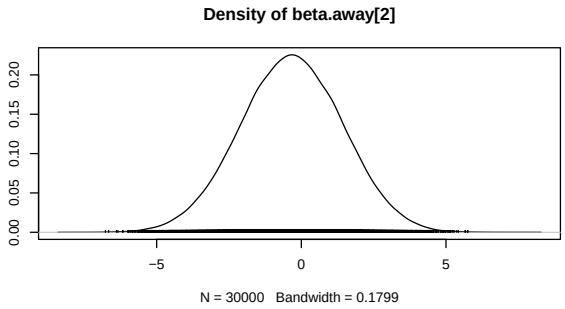
```
## Potential scale reduction factors:
##
##                 Point est. Upper C.I.
## beta.away[1]             1          1
## beta.away[2]             1          1
## beta.away[3]             1          1
## beta.away[4]             1          1
## beta.away[5]             1          1
## beta.away[6]             1          1
## beta.away[7]             1          1
## beta.away[8]             1          1
## beta.defense[1]          1          1
## beta.defense[2]          1          1
## beta.home[1]             1          1
## beta.home[2]             1          1
## beta.home[3]             1          1
## beta.home[4]             1          1
## beta.home[5]             1          1
## beta.home[6]             1          1
## beta.home[7]             1          1
## beta.home[8]             1          1
## delta[1]                 1          1
## delta[2]                 1          1
## eta[1]                   1          1
## eta[2]                   1          1
## eta[3]                   1          1
## eta[4]                   1          1
## rho[1]                   1          1
## rho[2]                   1          1
## rho[3]                   1          1
## rho[4]                   1          1
## sigmasq                  1          1
```

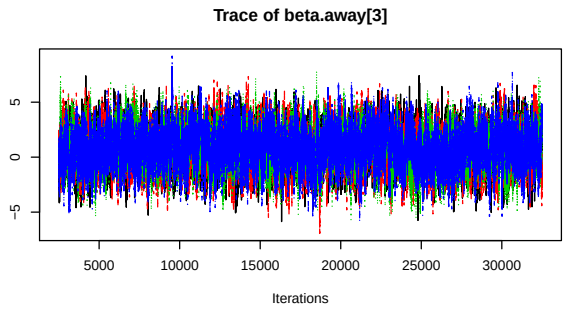
```
plot(result$coda.sam, smooth=FALSE)
```

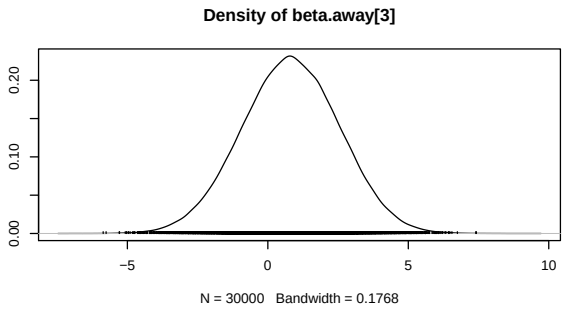

Trace of beta.away[1]
5
0
−5
5000
10000
15000
20000
25000
30000
Iterations

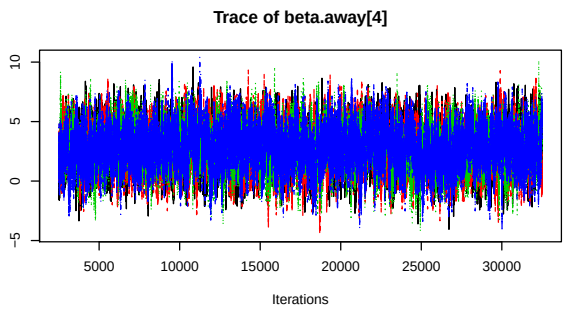

Density of beta.away[1]
0.20
0.10
0.00
−5
0
5
N = 30000 Bandwidth = 0.1706

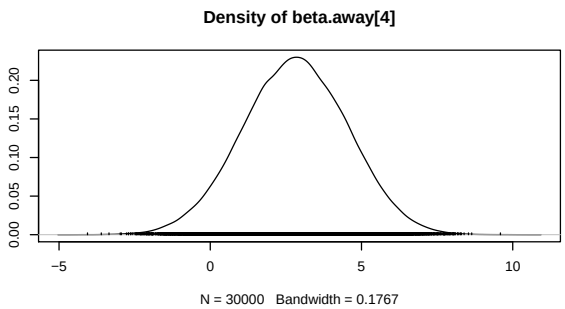

Trace of beta.away[2]
5
0
−5
5000
10000
15000
20000
25000
30000
Iterations

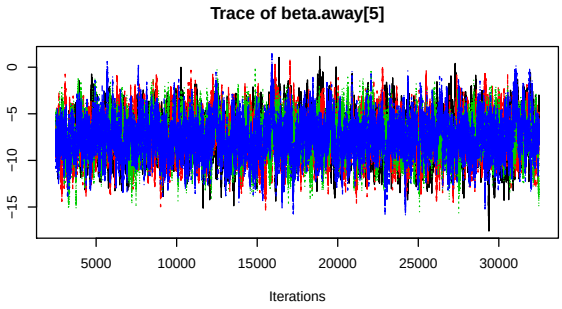

Density of beta.away[2]
0.20
0.15
0.10
0.05
0.00
−5
0
5
N = 30000 Bandwidth = 0.1799

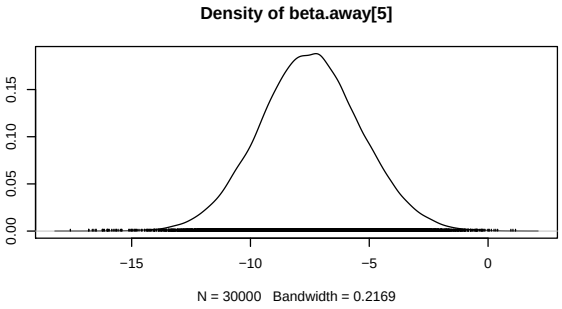

Trace of beta.away[3]
5
0
−5
5000
10000
15000
20000
25000
30000
Iterations

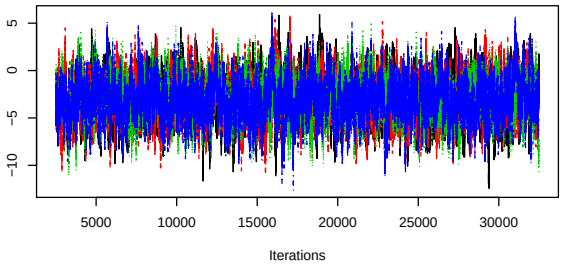

Density of beta.away[3]
0.20
0.10
0.00
−5
0
5
10
N = 30000 Bandwidth = 0.1768

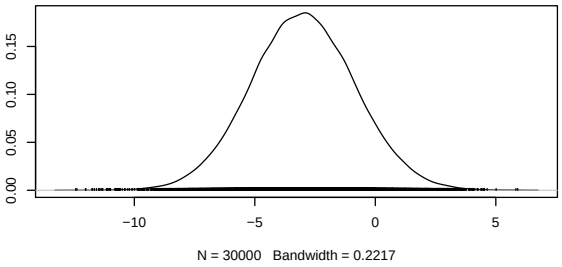

Trace of beta.away[4]
10
5
0
−5
5000
10000
15000
20000
25000
30000
Iterations

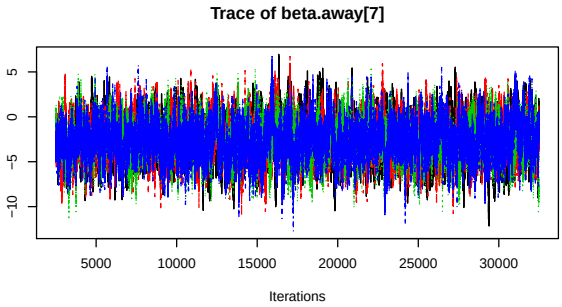

Density of beta.away[4]
0.20
0.15
0.10
0.05
0.00
−5
0
5
10
N = 30000 Bandwidth = 0.1767

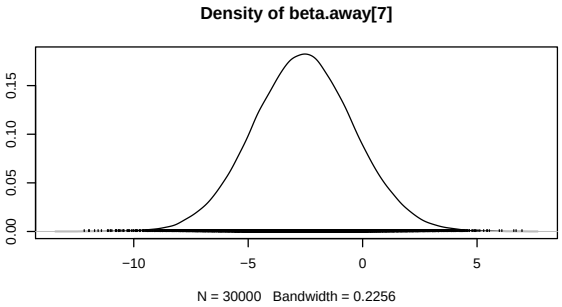

Trace of beta.away[5]
0
−5
−10
−15
5000
10000
15000
20000
25000
30000
Iterations

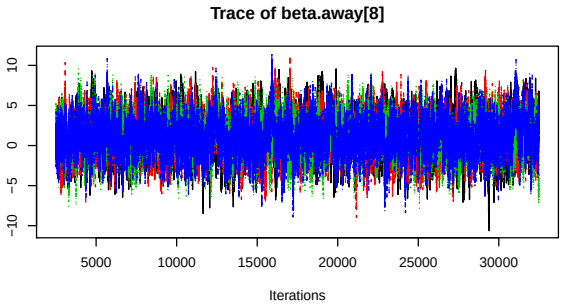

Density of beta.away[5]
0.00
0.05
0.10
0.15
−15
−10
−5
0
N = 30000 Bandwidth = 0.2169

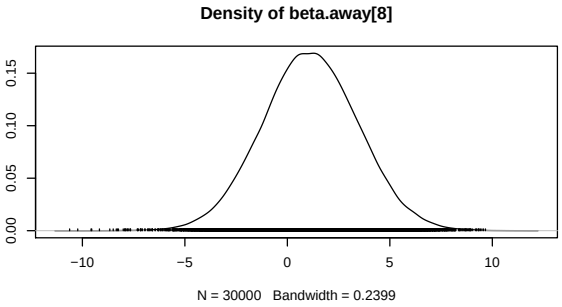

Trace of beta.away[6]
5
0
−5
−10
5000
10000
15000
20000
25000
30000
Iterations

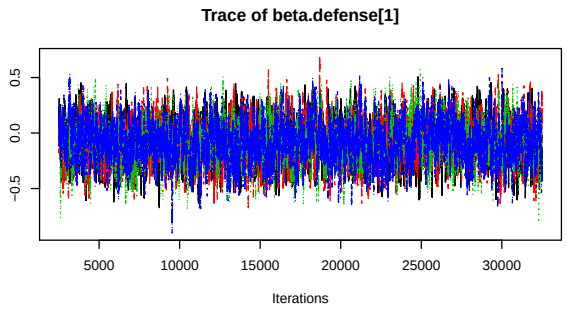

Density of beta.away[6]
0.00
0.05
0.10
0.15
−10
−5
0
5
N = 30000 Bandwidth = 0.2217

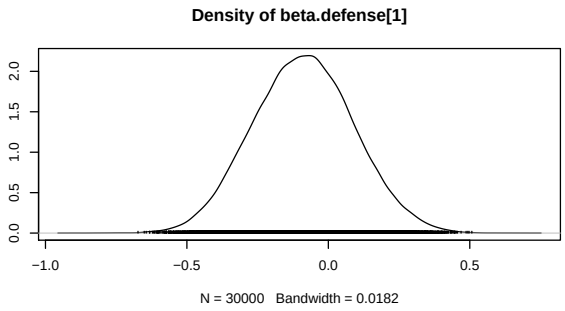

Trace of beta.away[7]
5
0
−5
−10
5000
10000
15000
20000
25000
30000
Iterations

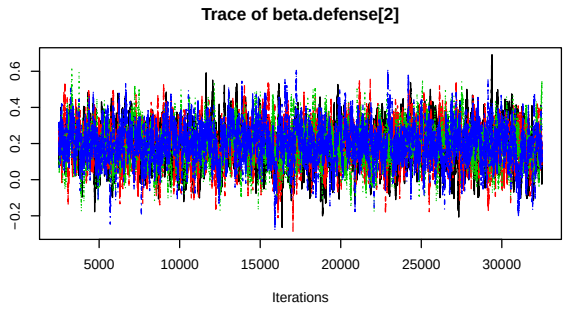

Density of beta.away[7]
0.00
0.05
0.10
0.15
−10
−5
0
5
N = 30000 Bandwidth = 0.2256

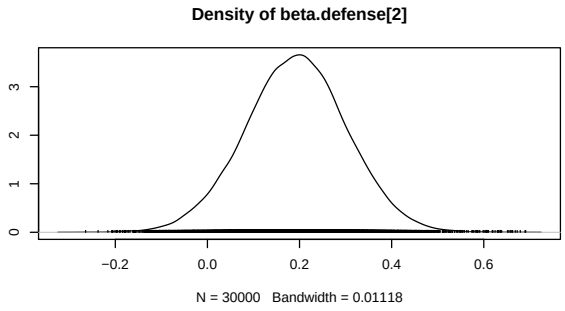

Trace of beta.away[8]
10
5
0
−5
−10
5000
10000
15000
20000
25000
30000
Iterations

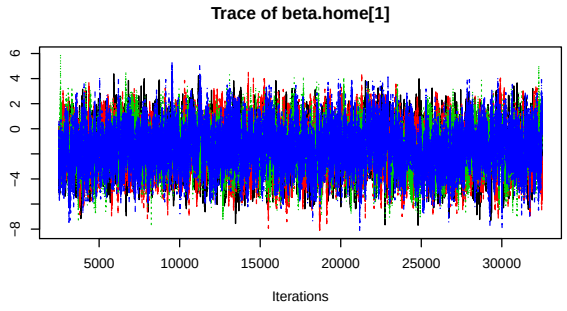

Density of beta.away[8]
0.00
0.05
0.10
0.15
−10
−5
0
5
10
N = 30000 Bandwidth = 0.2399

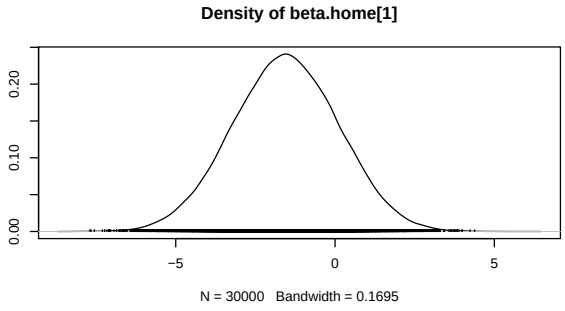

Trace of beta.defense[1]
0.5
0.0
−0.5
5000
10000
15000
20000
25000
30000
Iterations

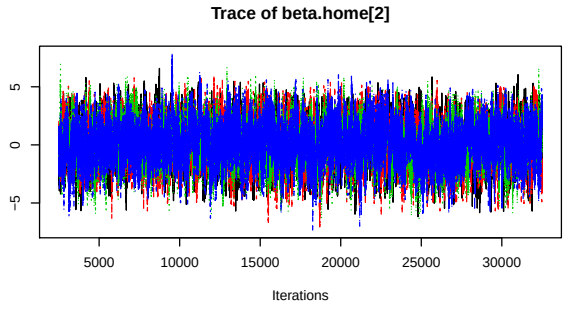

Density of beta.defense[1]
2.0
1.5
1.0
0.5
0.0
−1.0
−0.5
0.0
0.5
N = 30000 Bandwidth = 0.0182

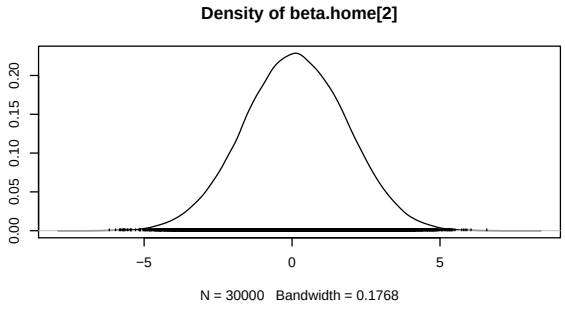

Trace of beta.defense[2]
0.6
0.4
0.2
0.0
−0.2
5000
10000
15000
20000
25000
30000
Iterations

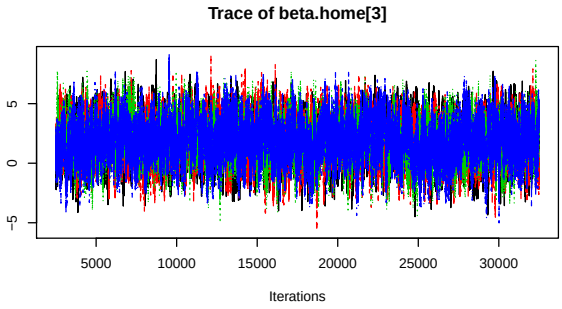

Density of beta.defense[2]
3
2
1
0
−0.2
0.0
0.2
0.4
0.6
N = 30000 Bandwidth = 0.01118

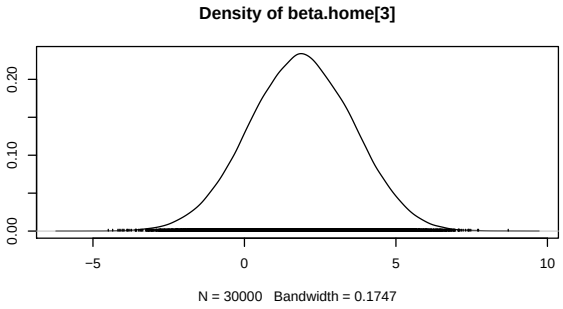

Trace of beta.home[1]
6
4
2
0
−4
−8
5000
10000
15000
20000
25000
30000
Iterations

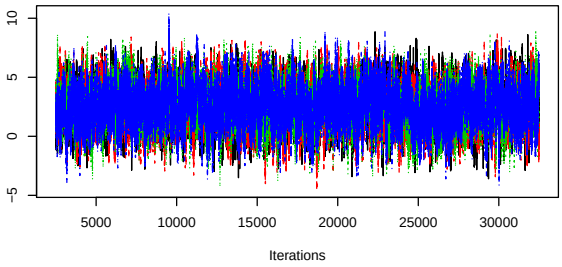

Density of beta.home[1]
0.20
0.10
0.00
−5
0
5
N = 30000 Bandwidth = 0.1695

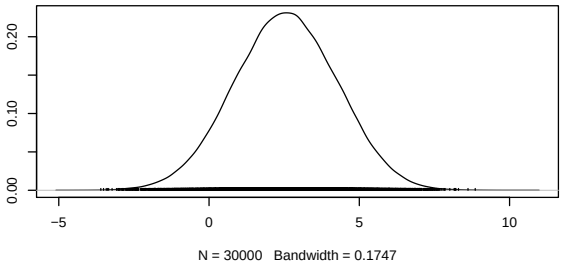

Trace of beta.home[2]
5
0
−5
5000
10000
15000
20000
25000
30000
Iterations

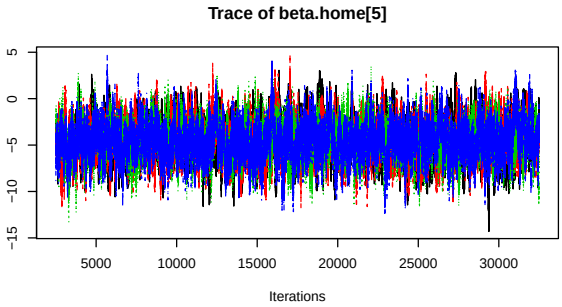

Density of beta.home[2]
0.20
0.15
0.10
0.05
0.00
−5
0
5
N = 30000 Bandwidth = 0.1768

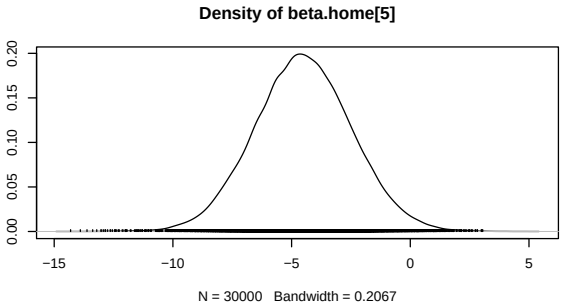

Trace of beta.home[3]
5
0
−5
5000
10000
15000
20000
25000
30000
Iterations

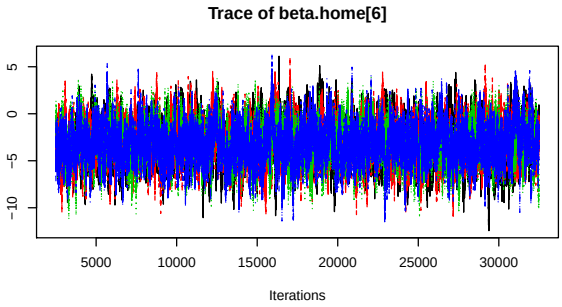

Density of beta.home[3]
0.20
0.10
0.00
−5
0
5
10
N = 30000 Bandwidth = 0.1747

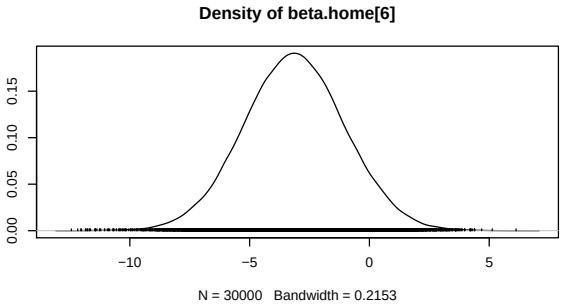

Trace of beta.home[4]
10
5
0
−5
5000
10000
15000
20000
25000
30000
Iterations

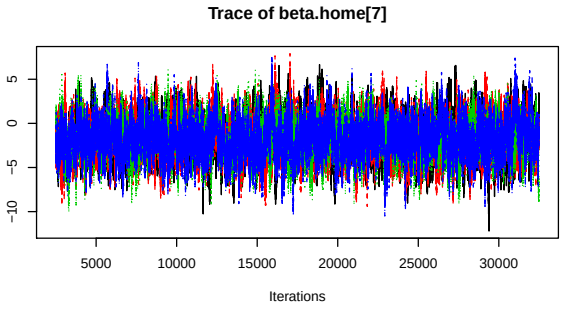

Density of beta.home[4]
0.20
0.10
0.00
−5
0
5
10
N = 30000 Bandwidth = 0.1747

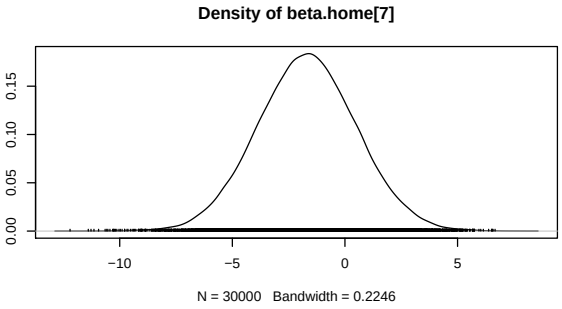

Trace of beta.home[5]
5
0
−5
−10
−15
5000
10000
15000
20000
25000
30000
Iterations

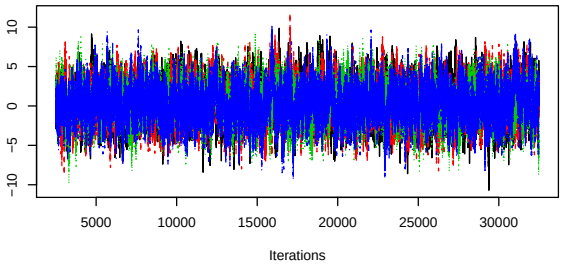

Density of beta.home[5]
0.20
0.15
0.10
0.05
0.00
−15
−10
−5
0
5
N = 30000 Bandwidth = 0.2067

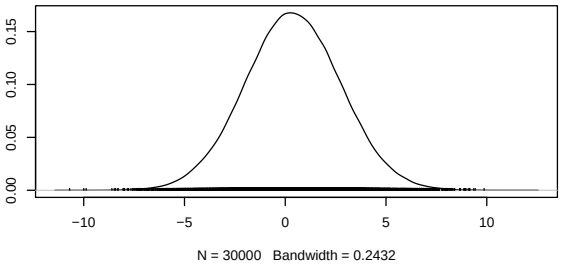

Trace of beta.home[6]
5
0
−5
−10
5000
10000
15000
20000
25000
30000
Iterations

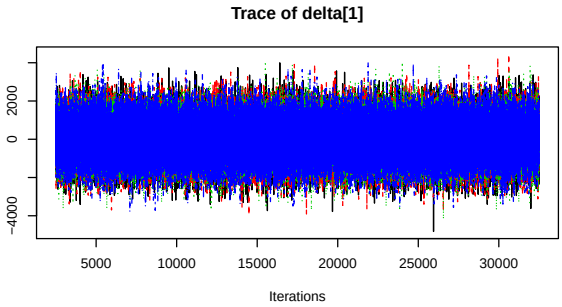

Density of beta.home[6]
0.15
0.10
0.05
0.00
−10
−5
0
5
N = 30000 Bandwidth = 0.2153

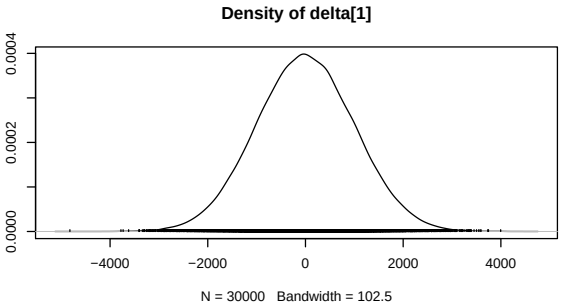

Trace of beta.home[7]
5
0
−5
−10
5000
10000
15000
20000
25000
30000
Iterations

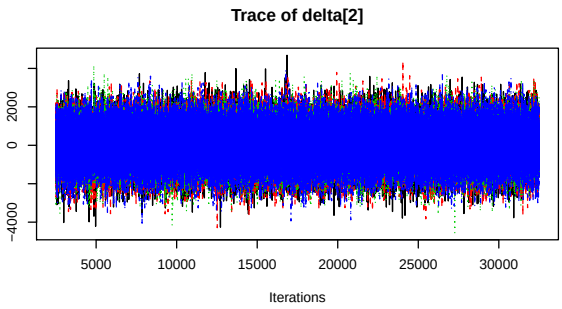

Density of beta.home[7]
0.15
0.10
0.05
0.00
−10
−5
0
5
N = 30000 Bandwidth = 0.2246

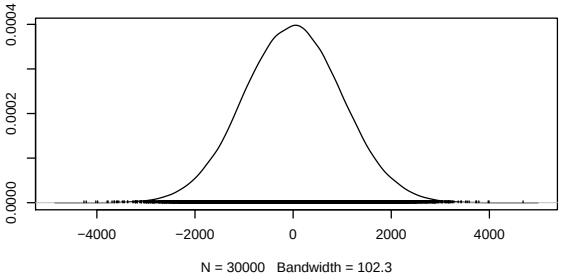

Trace of beta.home[8]
10
5
0
−5
−10
5000
10000
15000
20000
25000
30000
Iterations

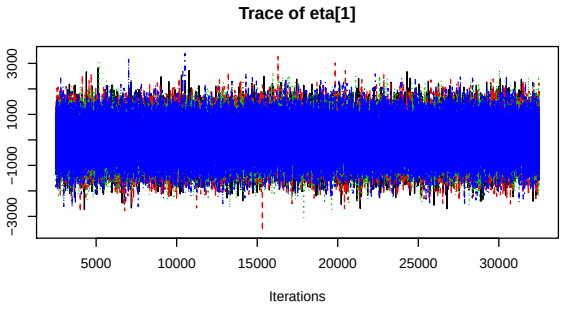

Density of beta.home[8]
0.15
0.10
0.05
0.00
−10
−5
0
5
10
N = 30000 Bandwidth = 0.2432

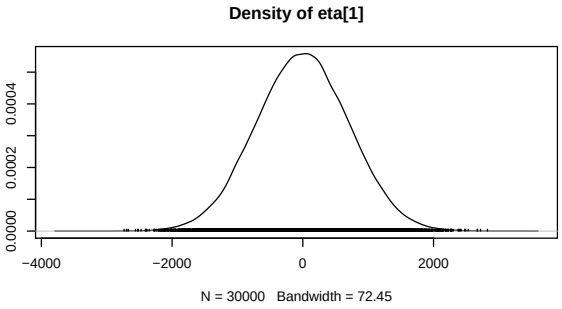

Trace of delta[1]
2000
0
−4000
5000
10000
15000
20000
25000
30000
Iterations

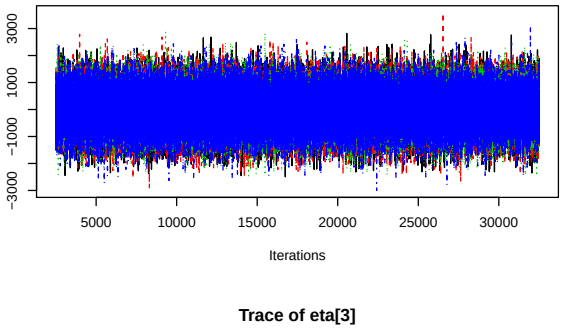

Density of delta[1]
0.0004
0.0002
0.0000
−4000
−2000
0
2000
4000
N = 30000 Bandwidth = 102.5

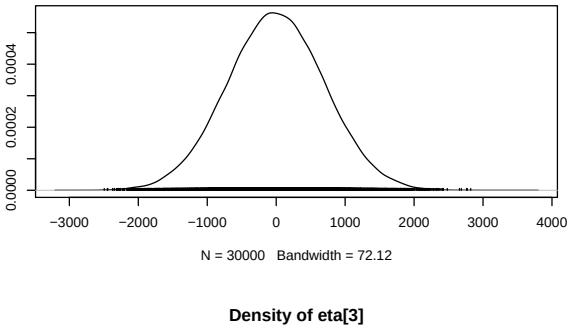

Trace of delta[2]
2000
0
−4000
5000
10000
15000
20000
25000
30000
Iterations

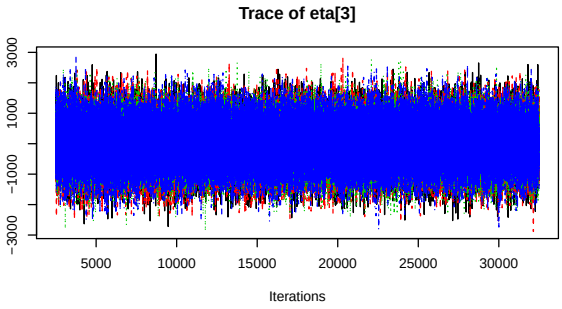

Density of delta[2]
0.0004
0.0002
0.0000
−4000
−2000
0
2000
4000
N = 30000 Bandwidth = 102.3

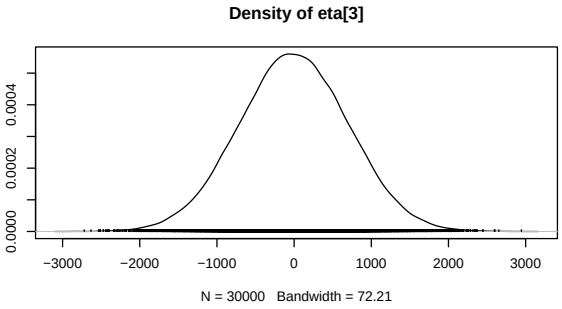

Trace of eta[1]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

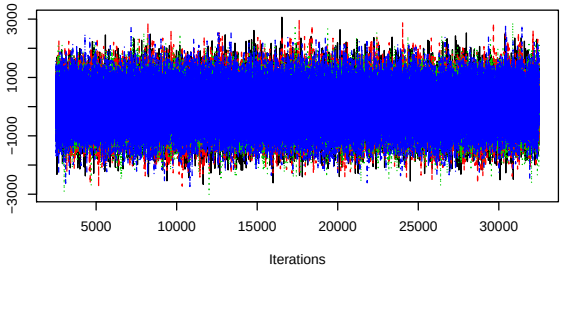

Density of eta[1]
0.0004
0.0002
0.0000
−4000
−2000
0
2000
N = 30000 Bandwidth = 72.45

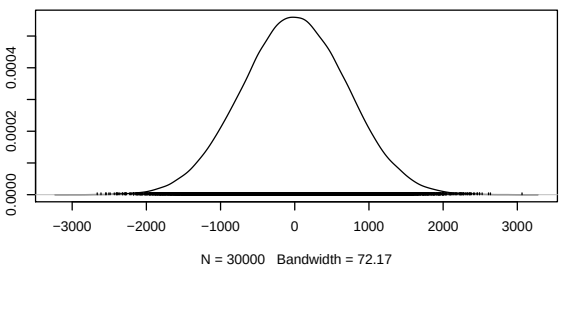

Trace of eta[2]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

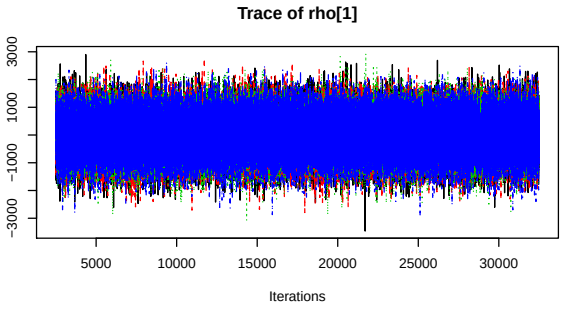

Density of eta[2]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
4000
N = 30000 Bandwidth = 72.12

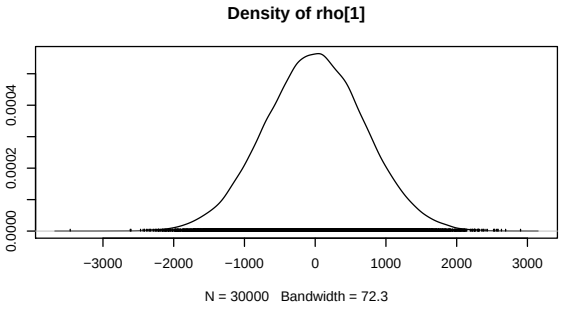

Trace of eta[3]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

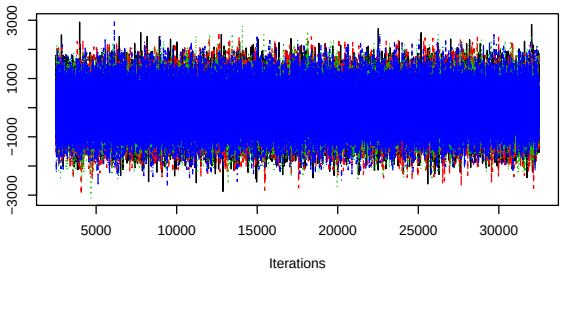

Density of eta[3]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.21

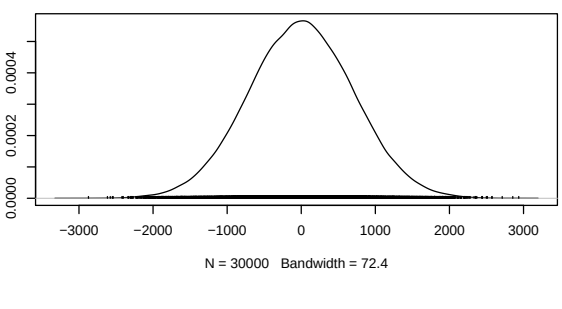

Trace of eta[4]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

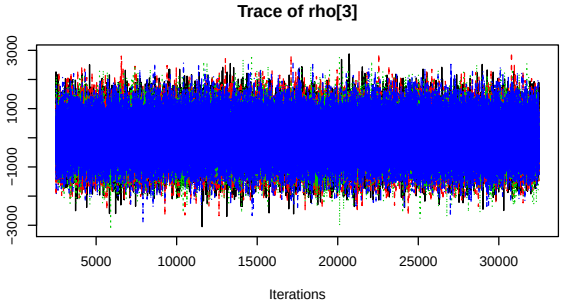

Density of eta[4]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.17

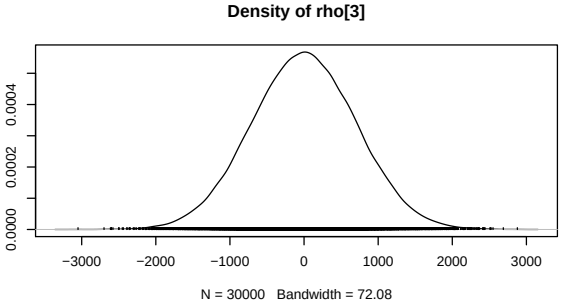

Trace of rho[1]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

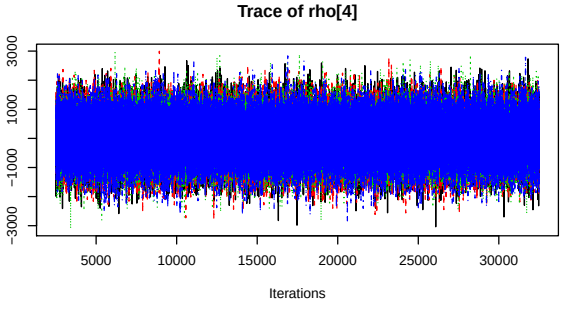

Density of rho[1]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.3

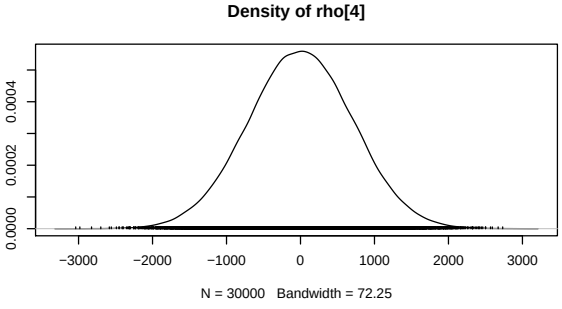

Trace of rho[2]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations

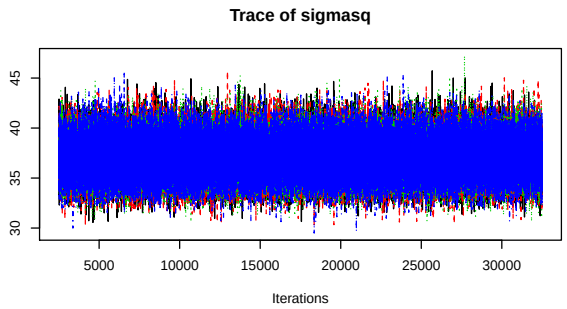

Density of rho[2]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.4

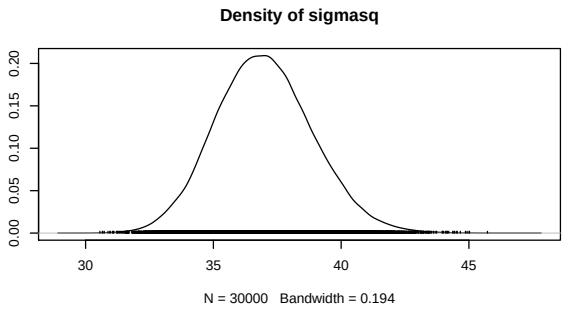

Trace of rho[3]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations


Density of rho[3]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.08


Trace of rho[4]
3000
1000
−1000
−3000
5000
10000
15000
20000
25000
30000
Iterations


Density of rho[4]
0.0004
0.0002
0.0000
−3000
−2000
−1000
0
1000
2000
3000
N = 30000 Bandwidth = 72.25





Converged as `gelman.R.max` = 1.0016 < 1.1 and the plot also looks good.

```
summary(result$coda.sam)
```

```
##
## Iterations = 2501:32500
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 30000
##
```

```
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                     Mean       SD Naive SE Time-series SE
## beta.away[1]    -1.6557    1.669 0.004818        0.03161
## beta.away[2]    -0.3388    1.760 0.005081        0.03284
## beta.away[3]     0.8113    1.730 0.004994        0.03214
## beta.away[4]     2.8111    1.729 0.004990        0.03202
## beta.away[5]    -7.4919    2.145 0.006191        0.04987
## beta.away[6]    -2.9995    2.169 0.006262        0.04964
## beta.away[7]    -2.5764    2.215 0.006394        0.04940
## beta.away[8]     1.1202    2.348 0.006778        0.04965
## beta.defense[1] -0.0877    0.178 0.000514        0.00376
## beta.defense[2]  0.1922    0.111 0.000319        0.00277
## beta.home[1]    -1.5498    1.659 0.004788        0.03111
## beta.home[2]     0.1104    1.730 0.004994        0.03221
## beta.home[3]     1.8742    1.710 0.004936        0.03203
## beta.home[4]     2.5501    1.709 0.004934        0.03227
## beta.home[5]    -4.5089    2.040 0.005890        0.04659
## beta.home[6]    -3.1311    2.118 0.006113        0.04818
## beta.home[7]    -1.6802    2.206 0.006368        0.05065
## beta.home[8]     0.3808    2.381 0.006874        0.05003
## delta[1]         0.2238 1002.758 2.894714        2.90766
## delta[2]        -0.3806 1000.678 2.888709        2.90938
## eta[1]          -4.3655  708.906 2.046434        2.04645
## eta[2]          -2.3423  705.636 2.036996        2.08072
## eta[3]          -4.0062  706.497 2.039481        2.05021
## eta[4]          -0.2471  706.171 2.038541        2.03855
## rho[1]          -2.8436  707.391 2.042061        2.05580
## rho[2]          -0.4743  708.381 2.044921        2.03798
## rho[3]          -0.2597  705.230 2.035823        2.02917
## rho[4]           1.1424  706.960 2.040818        2.03515
## sigmasq         36.9932    1.898 0.005480        0.00570
##
## 2. Quantiles for each variable:
##
##                      2.5%      25%     50%      75%    97.5%
## beta.away[1]       -4.9369   -2.784 -1.6497  -0.5174    1.579
## beta.away[2]       -3.8016   -1.530 -0.3343   0.8635    3.083
## beta.away[3]       -2.6015   -0.353  0.8148   1.9867    4.170
## beta.away[4]       -0.5980    1.642  2.8172   3.9941    6.166
## beta.away[5]      -11.6891   -8.921 -7.5017  -6.0769   -3.245
## beta.away[6]       -7.2386   -4.460 -3.0076  -1.5458    1.286
## beta.away[7]       -6.9259   -4.062 -2.5827  -1.1042    1.810
## beta.away[8]       -3.4653   -0.451  1.1128   2.6940    5.745
## beta.defense[1]    -0.4313   -0.210 -0.0877   0.0332    0.264
## beta.defense[2]    -0.0273    0.119  0.1930   0.2657    0.408
## beta.home[1]       -4.8030   -2.676 -1.5490  -0.4221    1.679
## beta.home[2]       -3.3014   -1.061  0.1114   1.2957    3.467
## beta.home[3]       -1.4932    0.714  1.8790   3.0470    5.182
## beta.home[4]       -0.8083    1.395  2.5523   3.7153    5.864
## beta.home[5]       -8.4899   -5.862 -4.5175  -3.1523   -0.477
## beta.home[6]       -7.2883   -4.547 -3.1364  -1.7234    1.029
## beta.home[7]       -5.9848   -3.156 -1.6833  -0.2105    2.681
```

```
## beta.home[8]       -4.2925   -1.210  0.3722   1.9793    5.076
## delta[1]        -1958.2972 -678.727 -2.6699 675.2980 1974.494
## delta[2]        -1961.3953 -677.259  0.5843 671.9379 1973.323
## eta[1]          -1393.0683 -484.262 -3.6522 475.2523 1379.799
## eta[2]          -1385.1205 -478.482 -3.6441 473.3419 1386.161
## eta[3]          -1391.9543 -478.354 -5.3446 474.3993 1377.874
## eta[4]          -1381.3403 -476.684 -0.9759 478.6431 1383.919
## rho[1]          -1393.3470 -478.265 -2.2613 473.4591 1386.487
## rho[2]          -1389.8083 -478.503  0.4387 477.5838 1388.661
## rho[3]          -1383.2185 -476.524 -0.8022 474.9497 1382.942
## rho[4]          -1390.6899 -474.040  1.7828 477.8879 1385.180
## sigmasq            33.4350   35.684 36.9349  38.2309   40.879
```

Effective Sample Size

```
(eff.size = effectiveSize(result$coda.sam[, ]))
```

```
##    beta.away[1]    beta.away[2]    beta.away[3]    beta.away[4]    beta.away[5]
##            2803            2887            2911            2927            1853
##    beta.away[6]    beta.away[7]    beta.away[8] beta.defense[1] beta.defense[2]
##            1915            2015            2241            2250            1602
##    beta.home[1]    beta.home[2]    beta.home[3]    beta.home[4]    beta.home[5]
##            2860            2906            2861            2824            1921
##    beta.home[6]    beta.home[7]    beta.home[8]        delta[1]        delta[2]
##            1934            1904            2270          118955          118315
##          eta[1]          eta[2]          eta[3]          eta[4]          rho[1]
##          120000          115553          118759          120000          118452
##          rho[2]          rho[3]          rho[4]         sigmasq
##          120833          120807          120680          111261
```

The effective sample sizes of all parameters are greater than 400.